Problem 8 (20 Points)

Problem Description

Several molecular dynamics simulations have been carried out for a material, and the phase (solid/liquid/vapor) at different temperature/pressure combinations has been recorded.

You will use gradient descent to train a One-vs-Rest logistic regression model on data with 3 classes. Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the previous problems.

Summary of deliverables:

- 3 binomial classification `w` vectors, corresponding to each class
- Function `classify(xy)` that evaluates all 3 models at a given array of points, returning the class prediction as the model with the highest probability
- Print model percent classification accuracy on the training data
- Print model percent classification accuracy on the testing data
- Plot that shows the training data as data points, along with the class of a grid of points in the background, as in the lecture activity.

Imports and Utility Functions:

In [117]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

def plot_data(x, y, c,title="Phase of simulated material"):
    xlim = [0,52.5]
    ylim = [0,1.05]
    markers = [dict(marker="o", color="royalblue"), dict(marker="s", color="crims
    labels = ["Solid", "Liquid", "Vapor"]

    plt.figure(dpi=150)

    for i in range(1+max(c)):
        plt.scatter(x[c==i], y[c==i], s=60, **(markers[i]), edgecolor="black", li

    plt.title(title)
    plt.legend(loc="upper right")
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel("Temperature, K")
    plt.ylabel("Pressure, atm")
    plt.box(True)

def plot_colors(classify, res=40):
    xlim = [0,52.5]
    ylim = [0,1.05]
    xvals = np.linspace(*xlim,res)
    yvals = np.linspace(*ylim,res)
    x,y = np.meshgrid(xvals,yvals)
    XY = np.concatenate((x.reshape(-1,1),y.reshape(-1,1)),axis=1)

    color = classify(XY).reshape(res,res)

    cmap = ListedColormap(["lightblue","lightcoral","palegreen"])
    plt.pcolor(x, y, color, shading="nearest", zorder=-1, cmap=cmap,vmin=0,vmax=2
    return
```

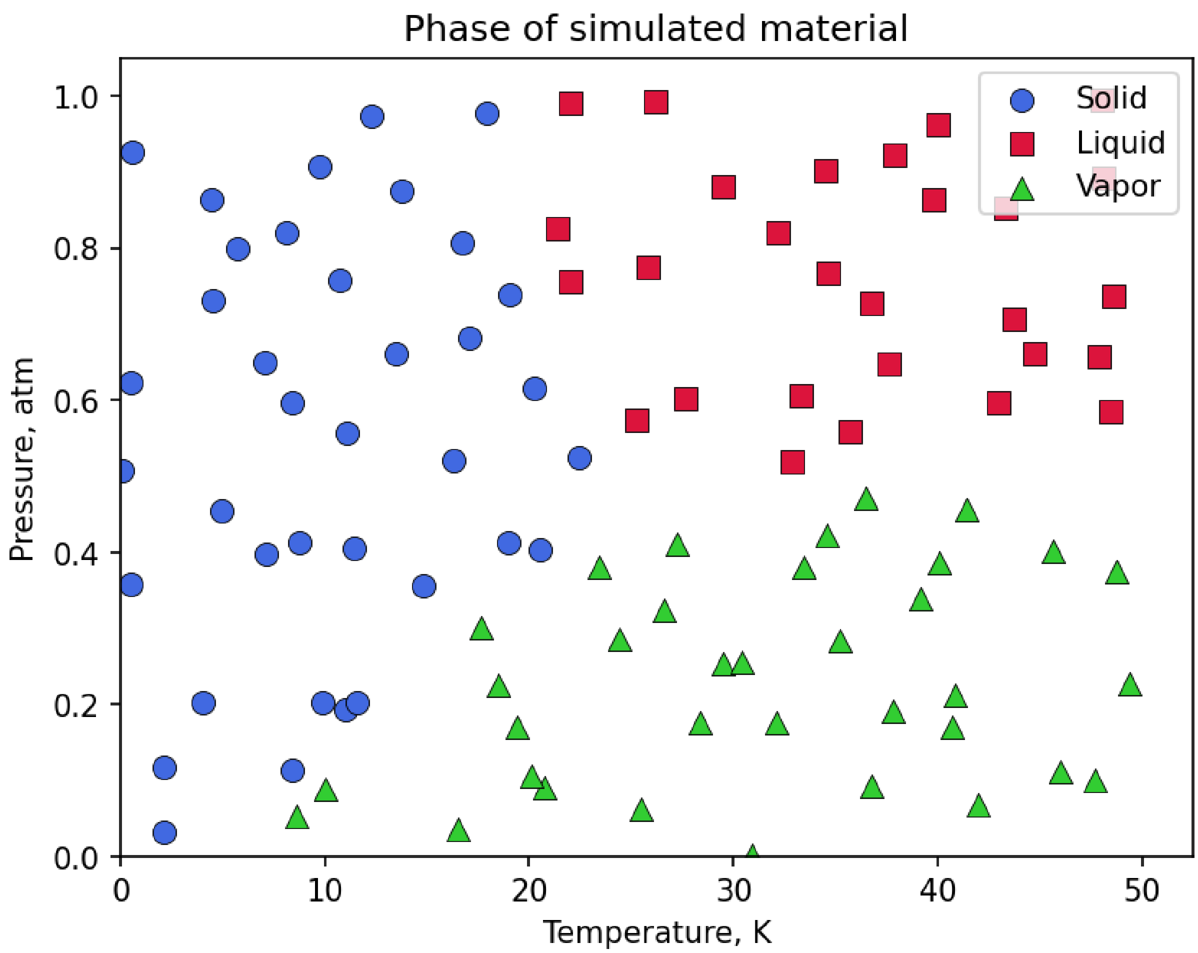
Load Data

This cell loads the dataset into the following variables:

- `train_data` : Nx2 array of input features, used for training
- `train_gt` : Array of ground-truth classes for each point in `train_data`
- `test_data` : Nx2 array of input features, used for testing
- `test_gt` : Array of ground-truth classes for each point in `test_data`

In the class arrays, 0 = solid, 1 = liquid, 2 = vapor.

In [118]:
```
train = np.load("data/w3-hw2-data-train.npy")
test = np.load("data/w3-hw2-data-test.npy")
train_data, train_gt = train[:,:2], train[:,2].astype(int)
test_data, test_gt = test[:,:2], test[:,2].astype(int)
plot_data(train_data[:,0], train_data[:,1],train_gt)
```



Gradient Descent

Here, write all of the necessary code to perform gradient descent and train 3 logistic regression models for a 1-vs-rest scenario. Use linear decision boundaries (features should only be 1, temperature, pressure)

Feel free to reuse code from the first problem or lecture activities.

We have provided the following function to help with the one-vs-all method:

`convert_to_binary_dataset(classes, A)` :

- Input: data, Nx2 array of temperature-pressure data
- Input: classes, array (size N) of class values for each point in `data`
- Input: A, the class (0, 1, or 2 here) to use as '1' in the binary dataset
- Returns: `classes_binary` , copy of classes where class A is 1, and all other classes are 0.

In [119]:
```
def convert_to_binary_dataset(classes, A):
    classes_binary = (classes == A).astype(int)
    return classes_binary
```

In [125]:
```
# YOUR CODE GOES HERE (gradient descent and related functions)

def sigmoid(h):
    return 1/(1+np.exp(-h))

def transform(data, w):

    xs = data[:,0]
    ys = data[:,1]
    ones = np.ones_like(xs)
    h = w[0]*ones + w[1]*xs + w[2]*ys

    return h

def loss(data, y, w):

    J1 = -np.log(sigmoid(transform(data,w))) * y
    J2 = -np.log(1-(sigmoid(transform(data,w)))) * (1-y)
    L = np.sum(J1 + J2)

    return L

def gradloss(data, y, w):

    column = np.ones((data.shape[0],1),dtype=int)
    data_t = np.hstack((column,data))
    grad = (sigmoid(transform(data,w)) - y) @ data_t

    return grad

def grad_desc(data, y, w0 = np.array([0,0,0]), iterations = 60000, stepsize=0.005

    for i in range(iterations):
        grad = gradloss(data,y,w0)
        w0 = w0 - stepsize*grad

    return w0
```

Training

Train your 3 models and print the `w` vector corresponding to each class

In [126]:
```
# YOUR CODE GOES HERE (training)

# converting it into binary dataset

class_0 = convert_to_binary_dataset(train_gt,0)
class_1 = convert_to_binary_dataset(train_gt,1)
class_2 = convert_to_binary_dataset(train_gt,2)

#getting w vectors

w0 = grad_desc(train_data,class_0)
w1 = grad_desc(train_data,class_1)
w2 = grad_desc(train_data,class_2)

# YOUR CODE GOES HERE (print "w"s)

print("w vector for class 0: ",w0)
print("w vector for class 1: ",w1)
print("w vector for class 2: ",w2)
```

```
w vector for class 0:  [19.13436556 -2.07871204 20.54839363]
w vector for class 1:  [-217.25859669    3.60389812  196.04458373]
w vector for class 2:  [   2.89835718    2.9215227  -219.24429555]
```

Classification function

Write a function `classify(xy)` that will evaluate each model and select the appropriate class.

In [127]:
```
def classify(xy):
    # YOUR CODE GOES HERE

    overall_pred = []

    pred_0 = sigmoid(transform(xy,w0))
    pred_1 = sigmoid(transform(xy,w1))
    pred_2 = sigmoid(transform(xy,w2))

    pred = np.vstack((pred_0,pred_1,pred_2)).T

    for i in range(xy.shape[0]):
        overall_pred.append(np.argmax(pred[i,:]))

    overall_pred = np.array(overall_pred)

    return overall_pred
```

Accuracy

Compute and print the accuracy on the training and testing sets as a percent

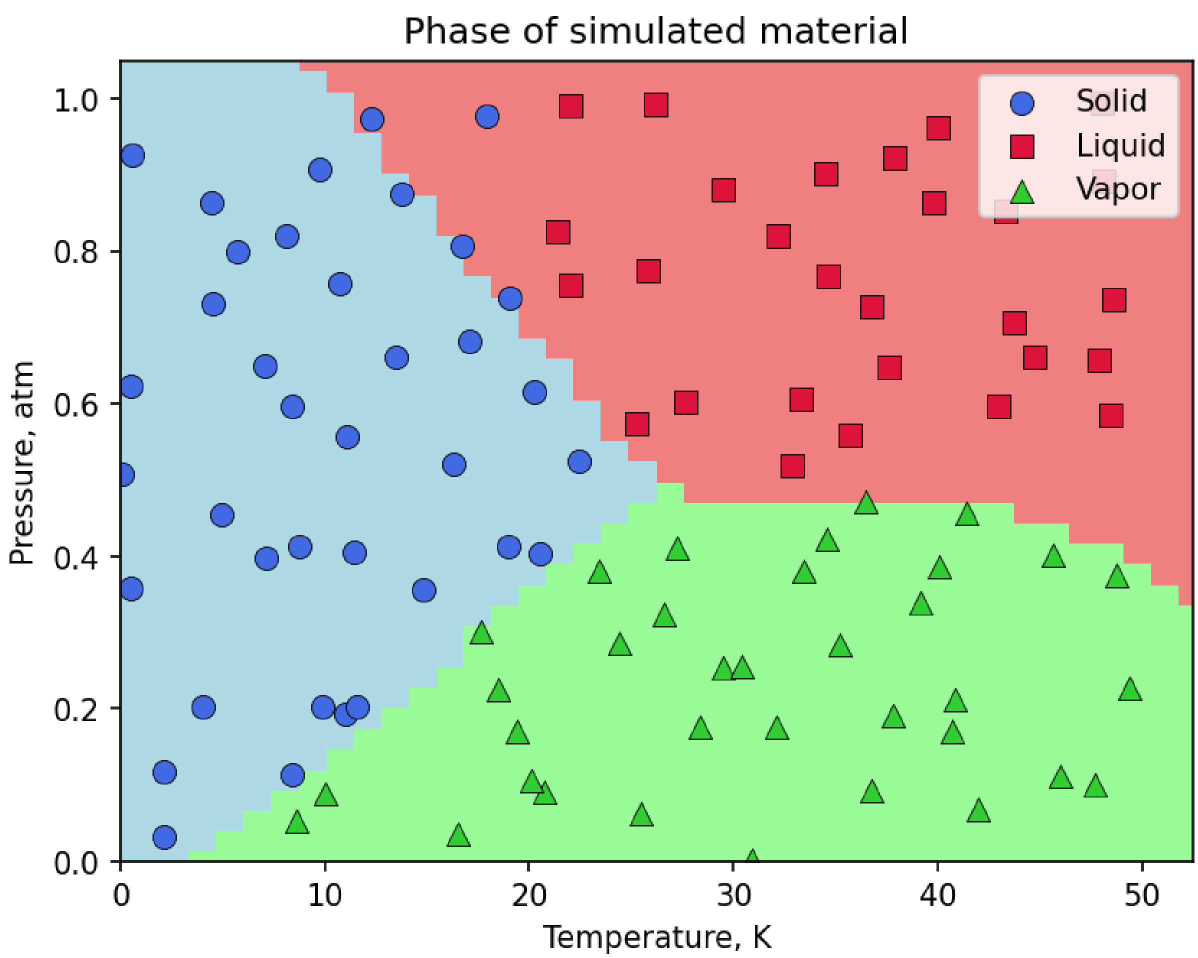
In [128]:
```
# YOUR CODE GOES HERE (accuracy)
prediction = classify(train_data)
accuracy = np.sum(prediction == train_gt) / len(train_gt) * 100
print("    Accuracy of training data: ", accuracy, r"%")

prediction = classify(test_data)
accuracy = np.sum(prediction == test_gt) / len(test_gt) * 100
print("    Accuracy of testing data: ", accuracy, r"%")
```

```
    Accuracy of training data:  94.0 %
    Accuracy of testing data:  96.0 %
```

Plot results

Run this cell to visualize the data along with the results of `classify()`

In [129]:
```
plot_data(train_data[:,0], train_data[:,1], train_gt)
plot_colors(classify)
```



In [ ]: